CS 475/575 -- Spring Quarter 2024

Project #5

CUDA: Monte Carlo Simulation

100 Points

Due: May 23

This page was last updated: April 15, 2024

Note: The flip machines do not have GPU cards in them, so CUDA and OpenCL will not run there. (You can compile there, you just can't run.) If your own system has a GPU, you can try using that. You can also use *rabbit* or the DGX machine.

Also, CUDA is an NVIDIA-only product. It will not run on Intel or AMD GPUs.

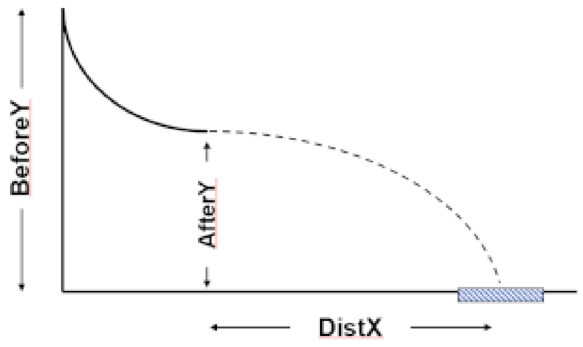
Introduction

Monte Carlo simulation is used to determine the range of outcomes for a series of parameters, each of which has a probability distribution showing how likely each option is to happen. In this project, you will take a scenario and develop a Monte Carlo simulation of it, determining how likely a particular output is to happen.

The Scenario

This project is inspired by this video. Be sure to watch it before reading on (it's only 41 seconds).

A golf ball is set in motion at the top of a ski jump at height *BeforeY*. It rolls until its height is *AfterY* at which point it shoots out horizontally. It continues until it hits the ground. A hole of radius RADIUS exists at a horizontal distance *DistX* from the end of the ski jump. Will the golf ball land inthe hole?



If we knew the exact measurements, we could figure this out exactly, but we don't have exact measurments. Several people have taken measurements and all have come up with different numbers. Their mean measurements and tolerances (±) are shown in the table below:

Quantity	Mean Value	± Value
BeforeY	70.f	5.f
AfterY	10.f	1.f

DistX	50.f	5.f
RADIUS	5.f	

Given all of this uncertainty of the dimensions, what is the *probability* that the golf ball will actually end up in the cup?

OK, you physics hounds, knock it off. Yes, we are going to neglect depth effects, moment of inertia, air resistance, etc, etc. Let's just have fun with this problem as stated.

Requirements:

- Run this for at least five BLOCKSIZEs (i.e., the number of threads-per-block) of 8, 32, 64, 128, and 256, combined with NUMTRIALS sizes of at least 1024, 4096, 16384, 65536, 262144, 1048576, and 2097152. You can use more if you want.
- Be sure each NUMTRIALS is a multiple of 1024. All of the ones above already are.
- Record timing for each combination. For performance, use some appropriate units like MegaTrials/Second.
- For this one, use CUDA timing, not OpenMP timing. It's already in the sample code.
- Produce a rectangular table and two graphs:
  1. Performance vs. NUMTRIALS with multiple colored curves of BLOCKSIZE
  2. Performance vs. BLOCKSIZE with multiple colored curves of NUMTRIALS
- Like Project #1 before, fill the arrays ahead of time with the random values. Send them to the GPU where they can be used as look-up tables. *Notice that the constant parameters are different here than in Project #1. Thus, your probability will be different.*
- Chosing one of the runs, tell me what you think the actual probability is.
- Parallel Fraction doesn't apply here, so don't compute one.

Developing this Project in Linux

Get the skeleton code here: proj05.cu
Left-click on it to see it.
Right-click on it to download it.

Here are .h files you will need (right-click to download each one):
exception.h
helper_functions.h
helper_cuda.h
helper_image.h
helper_string.h
helper_timer.h

On *rabbit* here is how to compile:

```
/usr/local/apps/cuda/cuda-10.1/bin/nvcc   -o proj05  proj05.cu
./proj05
```

On the DGX system, here is how to compile:

```
/usr/local/apps/cuda/11.7/bin/nvcc   -o proj05  proj05.cu
./proj05
```

For *rabbit*, here is a working Makefile:

```
proj05:         proj05.cu
                /usr/local/apps/cuda/cuda-10.1/bin/nvcc  -o proj05  proj05.cu
```

For *rabbit*, here is a working bash script:

```
#!/bin/bash
for t in 1024 4096 16384 65536 262144 1048576 2097152
do
        for b in 8 32 64 128 256
        do
                /usr/local/apps/cuda/cuda-10.1/bin/nvcc -DNUMTRIALS=$t -DBLOCKSIZE=$b -o proj05  proj05.cu
                ./proj05
        done
done
```

You can (and should!) write scripts to run the benchmark combinations. If you want to pass in benchmark parameters, the way you did it in g++ (**-DNUMTRIALS=$t**) notation works fine in nvcc.

Before you use the DGX, do your preliminary development on the *rabbit* system. It is a lot friendlier because you don't have to run your program through a batch submission. If you have time, take your final performance numbers on the DGX! They will be wonderfully higher than what you got on rabbit.

You can also take your benchmark numbers on your own machine.

numSuccesses++ Doesn't Work Anymore

In Project #1, you counted the number of times the golf ball went in the hole by saying **numSuccesses++**. In Project #5, there is no single numSuccesses on the GPU. So, now you will keep an array of successes in GPU memory. Your kernel will put either a 0 or a 1 there, and then your CPU program will add them all up.

Developing this Project in Visual Studio Enterprise 2022

Right-click on this link: VS05.zip, and save it somewhere. Then, un-zip it, and double-click on the **.sln** file. The skeleton code is in the file **kernel.cu** (that is what Visual STudio likes to call it when it creates it).

If you are trying to run CUDA on your own Visual Studio system, make sure your machine has the CUDA toolkit installed. It is available here: https://developer.nvidia.com/cuda- downloads?target_os=Windows&target_arch=x86_64&target_version=11&target_type=exe_local

Remember that CUDA is an NVIDIA-only product. It will not run on Intel or AMD GPUs.

Your PDF Commentary

Your commentary PDF should:

1. Tell what machine you ran this on
2. What do you think this new probability is?
3. Show the rectangular table and the two graphs
4. What patterns are you seeing in the performance curves?
5. Why do you think the patterns look this way?
6. Why is a BLOCKSIZE of 8 so much worse than the others?
7. How do these performance results compare with what you got in Project #1? Why?
8. What does this mean for what you can do with GPU parallel computing?

Grading:

Feature	Points
Correct probability	10
Monte Carlo performance table	20
Graph of performance vs. NUMTRIALS with multiple curves of BLOCKSIZE	20
Graph of performance vs. BLOCKSIZE with multiple curves of NUMTRIALS	20
Commentary -- explain the trends of the curves	30

Potential Total	**100**